International Journal of Neural Systems, Vol. 0, No. 0 (August, 2011) 00–00

SPAN: Spike Pattern Association Neuron for Learning Spatio-Temporal Sequences

Ammar Mohemmed, Stefan Schliebs
Knowledge Engineering and Discovery Research Institute
Auckland University of Technology, New Zealand
E-mail: amohemme@aut.ac.nz, sschlieb@aut.ac.nz

SATOSHI MATSUDA
Department of Mathematical Information Engineering
Nihon University, Japan
E-mail: matsuda.satoshi@nihon-u.ac.jp

NIKOLA KASABOV
Knowledge Engineering and Discovery Research Institute
Auckland University of Technology, New Zealand
and Institute for Neuroinformatics
ETH and University of Zurich, Switzerland
E-mail: nkasabov@aut.ac.nz

Received (to be inserted
Revised by Publisher)

Spiking Neural Networks (SNN) were shown to be suitable tools for the processing of spatio-temporal information. However, due to their inherent complexity, the formulation of efficient supervised learning algorithms for SNN is difficult and remains an important problem in the research area. This article presents SPAN – a spiking neuron that is able to learn associations of arbitrary spike trains in a supervised fashion allowing the processing of spatio-temporal information encoded in the precise timing of spikes. The idea of the proposed algorithm is to transform spike trains during the learning phase into analog signals so that common mathematical operations can be performed on them. Using this conversion, it is possible to apply the well-known Widrow-Hoff rule directly to the transformed spike trains in order to adjust the synaptic weights and to achieve a desired input/output spike behavior of the neuron. In the here presented experimental analysis, the proposed learning algorithm is evaluated regarding its learning capabilities, its memory capacity, its robustness to noisy stimuli and its classification performance. Differences and similarities of SPAN regarding two related algorithms, ReSuMe and Chronotron, are discussed.

1. Introduction

Spiking Neural Network (SNN) [1,2,3,4,5] advances the artificial computation paradigm by comprising temporal computational elements, the spiking neurons, that are biologically more plausible than the traditional neural models such as the perceptron. The synaptic connections between individual spiking neurons can be carefully modelled in order to exhibit a rich dynamic behavior with the intention to mimic the processes observed in biological synapses [6,7]. The salient feature of SNN is the neural code which is principally based on encoding the information into

stereotypical action potentials called spikes in analogy to the biological brain. However, the way this information is encoded in the brain is not well known.

Two main approaches are debated; rate coding and temporal coding. The rate coding paradigm assumes that the information is represented in the number of spikes that occur over a certain time period [1]. Temporal coding, on the other hand, encodes the information in the exact timing of individual spikes. Substantial empirical evidence supports the existence of such an encoding in neurology [8,9,10,11]. For example, in [12] it is shown that the large memory capacity of the brain is mostly attributed to the precise spike-timing nature of neural processing. Furthermore, some pattern recognition tasks such as the recognition of colors, visual patterns, odors and sound qualities can not be easily solved by rate-based neural models [13]. Temporal information encoding can also reduce the number of neurons that are necessary to perform a given task [14]. Supported by these observations and facts, this study will focus on the exact timing of spikes as the information encoding principle.

SNN has been employed mainly in neuroscience as a testbed to model various phenomenons exhibited in the brain in order to better understand the neural information processing and to get insights into brain related disorders [15,16,17]. In addition, SNN has proved itself adequate for a number of complex engineering problems [18,19,20,21,22,23,24,25]. It is considered a suitable computational tool to perform temporal pattern recognition and real-time computation [26,27,28]. However, due to its complexity and due to the fact that computing with spike trains is not straightforward, a number of challenges arise when using SNN in engineering applications. For example, it is not possible to add, subtract or multiply spike trains directly as it is possible with analog signals. Consequently, computing an error signal, a common operation employed in many supervised learning algorithms, can not be performed easily in SNN. The supervised training of SNN remains an important problem for the neural network research community.

SpikeProp [29] [14] is one of the first supervised learning algorithms for SNN. It was mainly employed for the processing of static (non-temporal) datasets. The method is based on a gradient descent on the error landscape analogous to the traditional backpropagation algorithm [30]. In SpikeProp, the error is defined as the temporal difference between a desired and an actual spike. The algorithm is applied to a multi-layer feedforward network in which the output neurons are trained to emit single spikes at desired firing times. These firing times are associated with a class label allowing the network to perform a classification problem. The algorithm is derived analytically and the discontinuities introduced by thresholding the membrane potential is circumvented by approximation.

It was shown that SpikeProp can solve nonlinear classification tasks such as the XOR, Iris, and Wisconsin classification problems [14]. The algorithm was modified in a number of studies. In [31] a momentum term was included in the update of the weights, while [32] extended the method to learn additional neural parameters, such as synaptic delays, time constants and neuron thresholds. An extension towards recurrent network topologies was presented in [33]. However, SpikeProp is able to deal with the firing of a single spike per neuron only. Therefore, it can not learn to produce multiple output spikes in response to a stimulus.

The Tempotron [34] enables a neuron to learn whether to fire or not to fire in response to a specific input stimulus. It implements a gradient descent dynamics that minimizes an error defined as the difference between the maximum membrane voltage generated by an erroneous pattern and the membrane firing threshold. Tempotron was evaluated to be efficient in binary temporal classification tasks.

It is worth to note that the above mentioned two methods are not able to learn reproducing spike trains. They are designed mainly for pattern recognition tasks, where the class of the pattern is identified by emitting or not emitting a single spike (Tempotron) or based on the timing of a single output spike (SpikeProp). However, in some applications, it is desired to reproduce a certain spike behavior, *i.e.* learning to generate a desired output pattern consisting of *multiple* spikes in response to a specific input stimulus.

ReSuMe [35,36] is one of the few algorithms to achieve this task efficiently. It is a supervised learning algorithm that is based on a learning window concept similar to the Spike Time Dependent Plasticity (STDP) [37,1,38]. The algorithm is described to be biologically plausible and can learn in an on-line fashion by adjusting the synaptic weights locally in both

space and time. ReSuMe interprets the Widrow-Hoff rule [39] as an interaction of two biological processes, *i.e.* an STDP (Hebbian) process and an anti-STDP (anti-Hebbian) process. The algorithm was shown to be efficient in a number of tasks including the learning of spike sequences and the classification of temporal spike patterns. When employed as the readout function of a Liquid State Machine (LSM) [26], the method is able to perform a mapping from any input spike train to any output spike train or even multiple output sequences. This capability of ReSuMe allows its application for parenthetic control systems [40].

A very recent supervised learning algorithm called Chronotron was proposed by Florian [41]. Similar to ReSuMe, it is also capable of learning spike sequence mappings using the precise timing of spikes. Two versions of the learning rule were proposed, an analytically-derived one, referred to E-learning, and one that is biologically plausible, referred to I-Learning. E-Learning is a gradient-descent optimization of the synaptic weights to minimize an error function defined as the difference between the actual output spike train and the desired spike train. The difference is measured using a modified version of the Victor & Purpura (VP) distance metric [42] that can handles the discontinuities inherent in this measure. The VP metric is one of the two metrics commonly used in neurobiology for quantifying the distance between two spike trains; the other is the van Rossum metric [43]. In contrast to the I-learning rule, E-learning implements an off-line learning process that requires the identification of the firing times of all spikes in the network in order to compute the error.

The other version of Chronotron, I-Learning, is similar to ReSuMe and can be used for on-line learning. In [41], an extensive analysis was undertaken to demonstrate the performance of the algorithm regarding its learning ability, its memory capacity, its learning behavior in the context of noisy input patterns and the effect of various parameters. The results show that E-learning, although being biologically less plausible, achieves a better performance in terms of the number of temporal patterns that can be learned compared to ReSuMe and I-learning.

In this paper, we propose another supervised learning algorithm for SNN that enables a single neuron to learn spike pattern associations. We refer to this learning neuron as SPAN for **S**pike **P**attern **A**ssociation **N**euron. In the SPAN learning algorithm, the input, output and desired spike trains are transformed into analog signals by convolving the spikes with a kernel function. This transformation will simplify the computation of the error signal and, hence, allows the application of a gradient descent to optimize the synaptic weights.

In [44], the authors used such a signal transformation along with a Particle Swarm Optimizer in order to optimize the parameters of dynamic synapses enabling the network to learn a desired input/output mapping of spike trains. However, due to scalability issues when training big networks, learning algorithms based on evolutionary computation are less practical. Therefore, a gradient descent method was suggested in [45]. Preliminary experiments were conducted demonstrating the functioning of the algorithm. In this study, we present a comprehensive analysis of the SPAN method along with a theoretical investigation highlighting the relationship of SPAN to ReSuMe and Chronotron.

The paper is organized as follows. In the next sections, we present the derivation of the SPAN learning rule followed by an experimental analysis of the learning capabilities, the memory capacity, the robustness and the classification performance of the proposed method. Finally, we discuss differences and similarities of SPAN regarding two related algorithms, namely ReSuMe and Chronotron, and derive conclusions of the presented study.

2. SPAN learning rule

Similar to other supervised training algorithms, the synaptic weights of a neuron are adjusted iteratively in order to impose a desired input/output spike sequence mapping. We derive the proposed learning algorithm from the common Widrow-Hoff rule, also known as the Delta rule. For a synapse i, it is defined as:

$$\Delta w_i = \lambda x_i \ (y_d - y_a) = \lambda x_i \delta_i \tag{1}$$

where $\lambda \in \mathbb{R}$ is a real-valued positive learning rate, x_i is the input transferred through synapse i, and y_d and y_a refer to the desired and the actual neural output, respectively. Note that $\delta_i = y_d - y_a$ is the difference or error between the desired and the actual output of the neuron.

This rule was introduced for traditional neural networks with linear neurons. For these models, the

input and output corresponds to real-valued vectors. In SNN however, trains of spikes are passed between neurons rendering the Widrow-Hoff rule not applicable to SNN. More specifically, if x_i, y_d and y_a were considered as spike trains $s(t)$ in the form of

$$s(t) = \sum_f \delta(t - t^f) \tag{2}$$

where t^f is the firing time of a spike and $\delta(\cdot)$ is the Dirac delta function $\delta(x) = 1$ if $x = 0$ and 0 otherwise, then the difference between two spike trains y_d and y_a does not define a suitable error landscape which can be minimized by a gradient descent.

In this paper, we address this issue by proposing the following idea. In order to define the difference between spike trains, we convolve each spike sequence with a kernel function $\kappa(t)$. This is similar to the binless distance metric used to compare spike trains [43]. We define

$$\tilde{x}_i(t) = \sum_{t_i^f \in F_{\text{in}}} \kappa(t - t_i^f) \tag{3}$$

$$\tilde{y}_d(t) = \sum_{t_d^g \in F_{\text{d}}} \kappa(t - t_d^g) \tag{4}$$

$$\tilde{y}_a(t) = \sum_{t_a^h \in F_{\text{a}}} \kappa(t - t_a^h) \tag{5}$$

with F_{in}, F_{d} and F_{a} being the input, the desired and the actual set of spikes, respectively. By substituting x_i, y_d and y_a with the kernelized spike trains $\tilde{x}_i(t)$, $\tilde{y}_d(t)$ and $\tilde{y}_a(t)$, a new learning rule for a spiking neuron is obtained:

$$\Delta w_i(t) = \lambda \tilde{x}_i(t)\ (\tilde{y}_d(t) - \tilde{y}_a(t)) \tag{6}$$

The equation formulates a real-time learning rule and so the synaptic weights change over time. By integrating Eq. 6, we derive the batch version of the learning rule which is under scrutiny in this paper:

$$\Delta w_i = \lambda \int_0^\infty \tilde{x}_i(t)\ (\tilde{y}_d(t) - \tilde{y}_a(t))\ \text{dt} \tag{7}$$

A variety of kernel functions $\kappa(t)$ exist such as linear, (double) exponential, alpha and Gaussian kernels. In this study, we use an α-kernel,$\alpha(t) = e\ \tau^{-1}\ t\ e^{-t/\tau} H(t)$, however many other kernels appear suitable in this context. A convolved spike train $\tilde{s}(t)$ is then given as

$$\tilde{s}(t) = \sum_{t^f} \kappa(t - t^f) \tag{8}$$

$$= \sum_{t^f} e\ \tau^{-1}\ (t - t^f)\ e^{-(t-t^f)/\tau} H(t - t^f) \tag{9}$$

where $H(t)$ refers to the Heaviside function and $\tau \in \mathbb{R}$ is a real-valued time constant. Using this kernel function, we can now perform the integration of Eq. 7:

$$\begin{aligned} \Delta w_i &= \lambda \int_0^\infty \Delta w_i(t)\ \text{dt} \\ &= \lambda \left(\frac{e}{2}\right)^2 \left[\sum_g \sum_f (|t_i^f - t_d^g| + \tau) e^{-\frac{|t_i^f - t_d^g|}{\tau}} \right. \\ &\quad \left. - \sum_h \sum_f (|t_i^f - t_a^h| + \tau) e^{-\frac{|t_i^f - t_a^h|}{\tau}} \right] \end{aligned} \tag{10}$$

With a simple example, the behavior of the presented learning rule can be demonstrated. Let us consider the case where the input, desired and actual spike trains have only a single spike at t_i, t_d, t_a, respectively and they satisfy $t_i \leq t_d \leq t_a$. Eq. 10 then simplifies to:

$$\begin{aligned} \Delta w_i &= \lambda \left(\frac{e}{2}\right)^2 \left[(t_d - t_i + \tau) e^{-\frac{t_d - t_i}{\tau}} \right. \\ &\quad \left. - (t_a - t_i + \tau) e^{-\frac{t_a - t_i}{\tau}} \right] \end{aligned} \tag{11}$$

and we note that

$$\Delta w_i \begin{cases} > 0 & \text{if} \quad t_d < t_a \\ = 0 & \text{if} \quad t_d = t_a \\ < 0 & \text{if} \quad t_a < t_d \end{cases} \tag{12}$$

From Eq. 12 several observations can be made:

- if the actual spike occurs later than the desired spike ($t_d < t_a$), then the synaptic weight increases and so the output spike will be emitted earlier at a next input presentation (epoch);
- conversely, if the actual spike occurs earlier than the desired firing time ($t_a < t_d$), then the synaptic weight decreases and so the output spike will be emitted later;
- if the actual spike occurs exactly at the desired time ($t_a = t_d$), then the synaptic weight does not change;

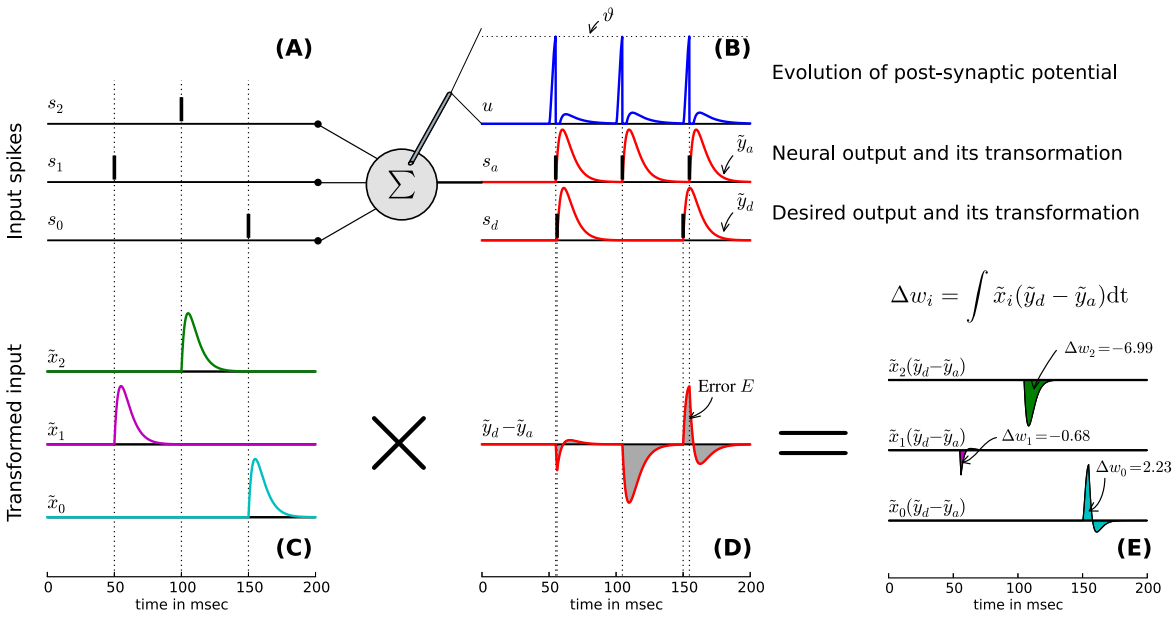


Figure 1: Illustration of the proposed learning rule SPAN. See text for detailed explanations of the figure.

- and, the larger the difference between t_a and t_d is, the larger the size of synaptic weight change becomes.

Furthermore, we can observe that

- when $t_a \to \infty$, which means that no actual spike occurs, the synaptic weight increases to promote the emission of an output spike since $t_d < t_a$ holds,
- when $t_d \to \infty$, which means no output spike is desired, the synaptic weight decreases to promote a suppression of an output spike since $t_a < t_d$ holds.

These observations are intuitively valid and we can expect, by repeating these processes, that the learning rule drives the post-synaptic neuron to emit a spike at the desired time. Furthermore, we note that the smaller the value of $t_d - t_i$ or $t_a - t_i$ is, the larger the value of each term in the square brackets of Eq. 11 becomes. That means that only if the input spike at t_i is temporally close to the desired or actual spike at t_d or t_a, *i.e.* spike t_i is the *cause* of spike t_d or spike t_a, the corresponding synaptic weight w_i changes significantly.

Weights are updated in an iterative process. In each iteration (or epoch), all input patterns are presented sequentially to the system. For each pattern the Δw_i values are computed and accumulated. After the presentation of all patterns, the weights are updated to $w_i(e+1) = w_i(e) + \Delta w_i$, where e is the current epoch of the learning process.

We note that the algorithm is capable of training the weights of a single neural layer only. Related methods such as ReSuMe [35] and the Chronotron [41] exhibit similar restrictions. Therefore, a combination with the well-known Liquid State Machine (LSM) approach [26] was suggested in these studies. By transforming the input into a higher-dimensional space, the output of the LSM can potentially be mapped to any desired spike train.

Figure 1 illustrates the functioning of the proposed SPAN learning method. An output neuron is connected to three input neurons through three excitatory synapses with randomly initialized weights. For the sake of simplicity, each input sequence consists of a single spike only. However, the learning method can also deal with more than one spike per input neuron. The inputs spike trains s_i are visualized in Figure 1A. In this example, we intend to train the output neuron to emit two desired spikes at the pre-defined time t_d^0 and t_d^1.

Assume that, as shown in Figure 1B, the presented stimulus causes the excitation of the output neuron resulting in the generation of three output spikes at times t_a^0, t_a^1 and t_a^2, respectively. Spike

t_a^0 is temporally very close to the desired spike t_d^0; spike t_a^1 is undesired and should be suppressed by the learning method; and spike t_a^2 occurs slightly too late ($t_d^1 < t_a^2$). The evolution of the membrane potential $u(t)$ measured at the output neuron is shown in middle top diagram of the figure above the actual and the desired spike trains, *cf.* Figure 1B.

The lower part in the figure (Figure 1C,D,E) depicts a graphical illustration of Equation 7. The input, actual and desired spikes trains are kernelized using the α-kernel as defined in Eq. 8 (Figure 1B and C). We define the area under the curve of the absolute difference $|y_d(t) - y_a(t)|$ as an error between actual and desired output:

$$E = \int_0^\infty |y_d(t) - y_a(t)| \text{ dt} \tag{13}$$

Although this error is not used in the computation of the weight updates Δw_i, this metric is an informative measure of the achieved training status of the output neuron.

Figure 1E shows the weight updates Δw_i. We especially note the large decrease of weight w_0. The input spike train s_0 of the first input neuron causes an undesired spike at t_a^1 and lowering the corresponding synaptic efficacy potentially suppresses this behavior. On the other hand, the synaptic weight w_2 is increased promoting the triggering of spike t_a^2 at an earlier time. Finally, weight w_1 remains almost unchanged since $t_a^1 \approx t_d^1$.

Unless otherwise stated, we use the batch version of the SPAN learning method along with the α-kernel for spike train convolution in the rest of this paper.

3. Experimental analysis of SPAN

In order to demonstrate the characteristics of the proposed learning algorithm, we have performed a number of computer simulations. First, we present a simple training scenario in which the mapping of a single input spike pattern to a single target spike train has to be learned. With this setup we verify the functioning of the learning algorithm and give further details on the training process.

In the second experiment, we investigate the robustness of the learning method. We mimic a real-world situation by adding a Gaussian noise to the input spike patterns which increases the difficulty of the learning task significantly.

The third experiment determines the memory capacity of SPAN, *i.e.* how many input spike patterns can be learned by the neuron. As already established in [34,41], the capacity of a neuron is dependent on the number of its synapses. Therefore, the presented experiment has a very practical background, since it provides an indication of how many synapses are necessary to perform a given learning task reliably.

Finally, in the fourth experiment, we apply SPAN on a classification task using a synthetic benchmark data set. In this experiment, we demonstrate the potential of SPAN for addressing practical real-world problems.

3.1. *Experimental setup*

For our experiments, we employ the Leaky Integrate-and-Fire (LIF) neuron which is one of the most widely used spiking neural models [1]. It is based on the idea of an electrical circuit containing a capacitor with capacitance C and a resistor with resistance R, where both C and R are assumed to be constant. The dynamics of a neuron i are then described by the following differential equation:

$$\tau_m \frac{du_i}{dt} = -u_i(t) + R\ I_i^{\text{syn}}(t) \tag{14}$$

The constant $\tau_m = RC$ is called the membrane time constant of the neuron. Whenever the membrane potential u_i crosses a threshold ϑ from below, the neuron fires a spike and its potential is reset to a reset potential u_r. Following [1], we define

$$t_i^{(f)} : u_i(t^{(f)}) = \vartheta, f \in \{0, \ldots, n-1\} \tag{15}$$

as the firing times of neuron i where n is the number of spikes emitted by neuron i. It is noteworthy that the shape of the spike itself is not explicitly described in the traditional LIF model. Only the firing times are considered to be relevant.

The synaptic current I_i^{syn} of neuron i is modeled using an α-kernel:

$$I_i^{\text{syn}}(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)}) \tag{16}$$

where $w_{ij} \in \mathbb{R}$ is the synaptic weight describing the strength of the connection between neuron i and its pre-synaptic neuron j. The α-kernel itself is defined as

$$\alpha(t) = e\ \tau_s^{-1}\ t\ e^{-t/\tau_s} H(t) \tag{17}$$

Model Summery	
Neural model	Leaky integrate-and-fire
Synaptic model	α shaped synaptic currents
Input	Random input
Connectivity	All input neurons are connected to a single output neuron
Neural Model	
Type	Leaky integrate-and-fire (LIF) neuron
Description	Dynamics of membrane potential $u(t)$: • Spike times: $t^{(f)} : u(t^{(f)}) = \vartheta$ • Sub-threshold dynamics: $\tau_m \frac{du}{dt} = -u(t) + R\ I^{\text{syn}}(t)$ • Reset & refractoriness: $u(t) = u_r \forall f : t \in (t^{(f)}, t^{(f)} + \tau_{\text{ref}})$ • exact integration with temporal resolution dt
Parameters	Membrane time constant $\tau_m = 10$ms Membrane resistance $R = 333.33\text{M}\Omega$ Spike threshold $\vartheta = 20$mV, reset potential $u_r = 0$mV Refractory period $\tau_{\text{ref}} = 3$ms Time resolution $dt = 0.1$ms, simulation time $T = 200$ms
Synaptic Model	
Type	Current synapses with $\alpha-$function shaped post-synaptic currents (PSCs)
Description	Synaptic input current $I^{\text{syn}}(t) = \sum w \sum_f \alpha(t - t^{(f)})$ $\alpha(t) = \begin{cases} e\ \tau_s^{-1}\ t\ e^{-t/\tau_s}, & \text{if } t > 0 \\ 0, & \text{otherwise} \end{cases}$
Parameters	Synaptic weight $w \in \mathbb{R}$, uniformly randomly initialized in $[0, 25]$ Synaptic time constant $\tau_s = 5$ms
Input Model	
Type	Random input
Details	Population of 200 input neurons each firing a single spike at a randomly chosen time in the period $(0, T)$

Table 1: Tabular description of the experimental setup as suggested in [46].

where $H(t)$ refers to the Heaviside function and parameter τ_s is the synaptic time constant.

We follow the initiative recently proposed in [46] that promotes reproducible descriptions of neural network models and experiments. The initiative suggests the use of specifically formatted tables explaining neural and synaptic models along with their parametrization. We use a similar setup in all of our experiments, *cf.* Table 1. In all experiments, the network architecture consists of single neuron driven by n synapses. The input spike patterns stimulating the neuron are generated randomly. More specifically, each input spike train consists of a single spike generated randomly in the time interval (0, 200 ms). The simulation is performed using the NEST simulator [47]. We provide the setup details that are specific for a particular experiment in the individual sections below.

3.2. *Sequence learning*

The purpose of the first experiment is to demonstrate the concept of the proposed learning method. The task is to learn a mapping from a random input spike pattern to specific target output spike train. This target consists of five spikes occurring at the times $t_d^0 = 33$, $t_d^1 = 66$, $t_d^2 = 99$, $t_d^3 = 132$ and $t_d^4 = 165$ms. Initially, the synaptic weights are randomly generated uniformly in the range $(0, 25\text{pA})$. Over 100 epochs, we allow the output neuron to adjust its connection weights in order to produce the desired output spike train. The experiment is repeated for 100 runs each of them initialized with different random weights in order to guarantee statistical significance.

In Figure 2, the experimental setup of a typical run is illustrated. The left side of the diagram shows the network architecture as defined in the experimental setup above. The right side shows the desired target spike train (top) along with the produced spike trains by the output neuron over a number of learning epochs (bottom). We note that the

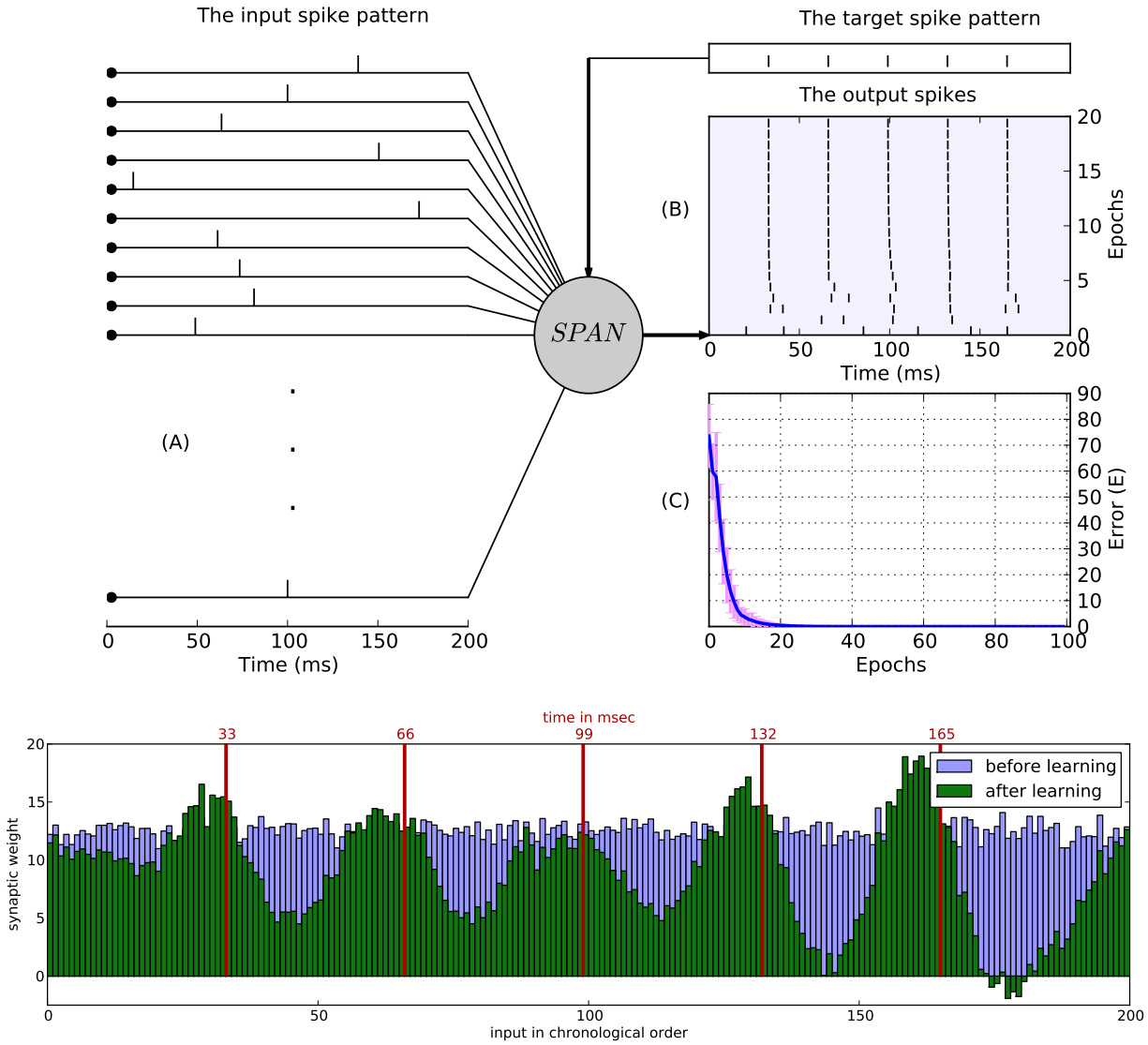


Figure 2: Learning spike pattern association with 400 input synapses. The neuron learns to map between spatio-temporal input pattern and output spike train. (B) The development of the output toward the target pattern for one of the trials. (C) The evolving of the error and standard deviation. (D) The synaptic weights before and after the learning process.

output spike trains in early epochs are very different from the desired target spike sequence. In later epochs the output spikes converge towards the desired sequence. Consequently, the error as defined in Equation 13 decreases in succeeding epochs. We note that the neuron is able to reproduce the desired spike output pattern very precisely in less than 20 learning epochs.

Figure 2C shows the evolution of the average error over the performed 100 runs. We note the exponential decrease of the error. In 97% of all trials the target spike train could be reproduced in less than 30 epochs and even for the remaining three percent, the average temporal difference between learned and desired spike train was less than 0.2 ms.

The effect of the learning algorithm on the synaptic efficacy can be visualized by comparing the synaptic weights before and after the application of the

learning process, *cf.* Figure 2D. For the diagram, the neural inputs are chronologically sorted according to their spike firing times. A bar in the figure reflects the synaptic strength of a synapse that corresponds to a particular input. In order to get an impression of the temporal causality of the weight changes, we overlay the plot with the desired firing times of the neuron (red vertical lines at 33, 66, 99, 132 and 166ms). The figure presents the weight changes averaged over all 100 runs.

Due to the experimental setup, we observe a uniform distribution of the weights after the initialization of the algorithm. After the training over 100 epochs, the synapses that transfer input spikes which are temporally close to the desired target spikes are potentiated. On the other hand, synapses that transfer spike inputs at undesired times are inhibited. The sine-shaped form of the chronologically sorted synaptic efficacies is caused by the equidistant firing times of the spikes in the target sequence.

From this simple experiment, we conclude that the proposed learning method is indeed able to train a desired input-output behavior to a spiking neuron. In the next sections, we investigate some more challenging learning scenarios for SPAN.

3.3. *Learning with noise*

The previous experiment involved the learning of a single pattern only. In this experiment, we investigate the performance of SPAN when several input patterns have to be learned. Furthermore, we are interested in the behavior of the method when the input stimuli are noisy which is important in the light of a real-world application.

Our experimental setup was inspired by the study presented in [41]. We construct an initial set of ten spike patterns each consisting of $n = 500$ input neurons that are allowed to emit a single spike only. With every presentation of an input pattern to the learning neuron, a noise is added to each spike in form of a jitter drawn from a Gaussian distribution. The strength of the jitter is controlled by the standard deviation of the Gaussian. In our experiments, we use different jitter strengths in order to investigate the impact of different noise levels on the learning performance of SPAN.

The neuron is trained in 400 epochs to emit a single spike at $t_d = 99$ ms in response to the input patterns. We call the output of the neuron successful, if the output sequence consists of a single spike only that occurs within the interval $[t_d - 5\text{ms}, t_d + 5\text{ms}]$. We define P_s as the probability of a successful output. It is the ratio of the number of output spikes that match their desired spikes over all ten input patterns. We consider jitter strengths of 0, 5, 10, 15 and 20ms. For each of them, an individual experiment is undertaken and repeated for 100 trials to guarantee statistical evidence.

Figure 3 presents the results of the experiment averaged over the 100 trials. The top row of diagrams show the obtained results for the noise-free case, *i.e.* a jitter strength of zero. On the left, the evolution of the error is presented. In the first few iterations of training, the neuron spikes arbitrary and the output does not match the desired target. We note that P_s (depicted in the right top diagram) is low in the first few epochs of the training process. However, the output stabilizes quickly and P_s increases rapidly indicating the neuron's ability to converge its output to the desired target spike.

In order to give an impression of the temporal difference between the obtained output spike and the target spike, we have computed the absolute difference $\Delta t = |t_d - t_a|$ for all successful output spikes. The evolution of Δt is overlaid in the right top diagram. Clearly, the temporal difference is minimized quickly by SPAN's learning algorithm resulting in very precisely timed output spikes.

If noise is introduced to the presented input patterns, the difficulty of the learning task increases significantly. The diagrams in the middle row of Figure 3 present the results for a jitter strength of 5ms. As expected, the training error can not become zero in this learning scenario. However, the evolution of the error indicates a certain convergence of the algorithm. Despite the noise, the training is very often successful. More than 90% of the output spikes fulfil the defined success criterion. The neuron is able to learn to fire within an average time shift $\Delta t = 2$ms irrespective of the noise.

The performance of SPAN as a function of the jitter strength is depicted in the bottom plots of Figure 3. For the diagrams we have used the neural outputs obtained during the last training epoch. Clearly, the error is proportional to the jitter strength. This relationship indicates a satisfying resistance of the SPAN rule to input pattern noise.

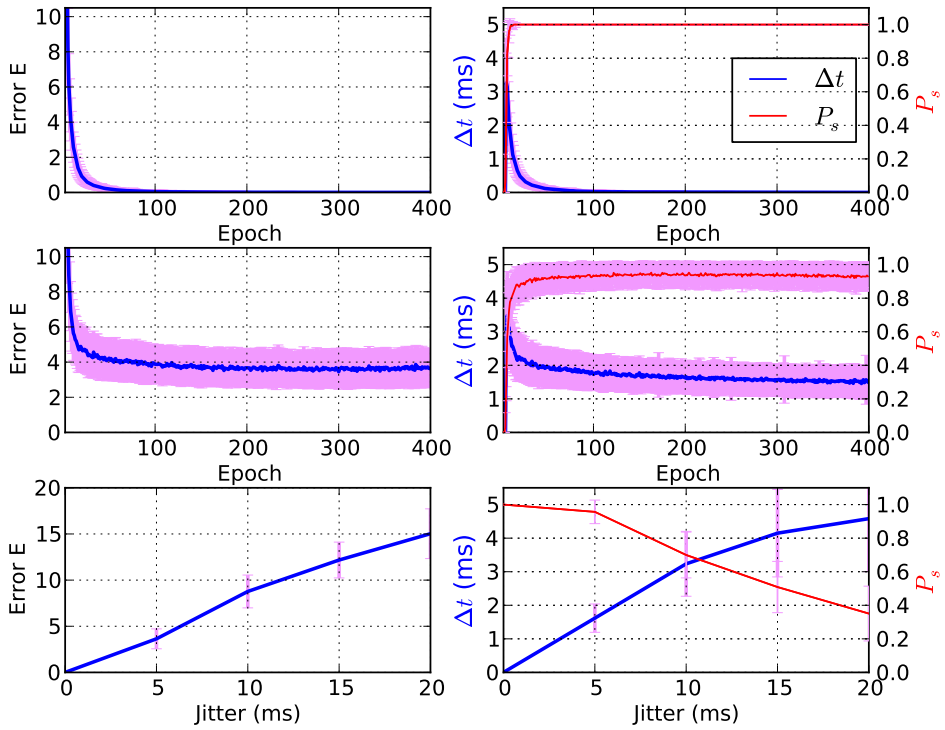


Figure 3: Learning multiple spike patterns using the SPAN learning rule. The top plots show the results when the patterns are learned without any noise applied. The diagrams below show the learning when jittered input patterns are used (jitter strength of 5ms). A neuron is trained to fire a single spike at 99 ms. The success probability P_s is computed in every epoch to indicate the number of times the output spikes matches the desired spike. The bottom diagrams show the final training error in dependence of the applied jitter strength.

Even for large jitter strengths, the method is able to map around three out of ten pattern correctly, *cf.* right bottom diagram of the figure.

3.4. *Memory capacity*

An important issue related in the learning process is how much information the neuron can learn and memorize. We use the measure proposed in [34] to evaluate the memory capacity of SPAN. The memory capacity is described in term of the load factor which is defined as the ratio of the number of input patterns p the neuron can classify correctly over the number of synapses n, *i.e.* $\frac{p}{n}$.

The p input patterns are generated randomly, similar to the previous experiments, where each pattern consists of n spike trains, each has a single spike at a random time instant. Subsequently, the patterns are assigned randomly to c different classes, which is set to 5 in this experiment.

The task of the experiment is to train the neuron to classify the all patterns correctly in a maximum number of epochs of 500. The classification is performed by training the neuron to fire a a single spike at a specified time instant t_d^i when a pattern that belongs to class i is presented at the input. Thus, the class of the input pattern is identified by the time of the fired spike, t_d^i, which is set to 33, 66, 99, 132, or 165 ms to identify the five classes.

The experiment is repeated on three network architectures having 200, 400 and 600 synapses. We report the success rate as a function of the number of the input patterns p. The success rate is the percentage of trials having the all input patterns classified successfully, also we report the average number of iterations required to achieve successful classification.

A pattern is decided as correctly classified if the

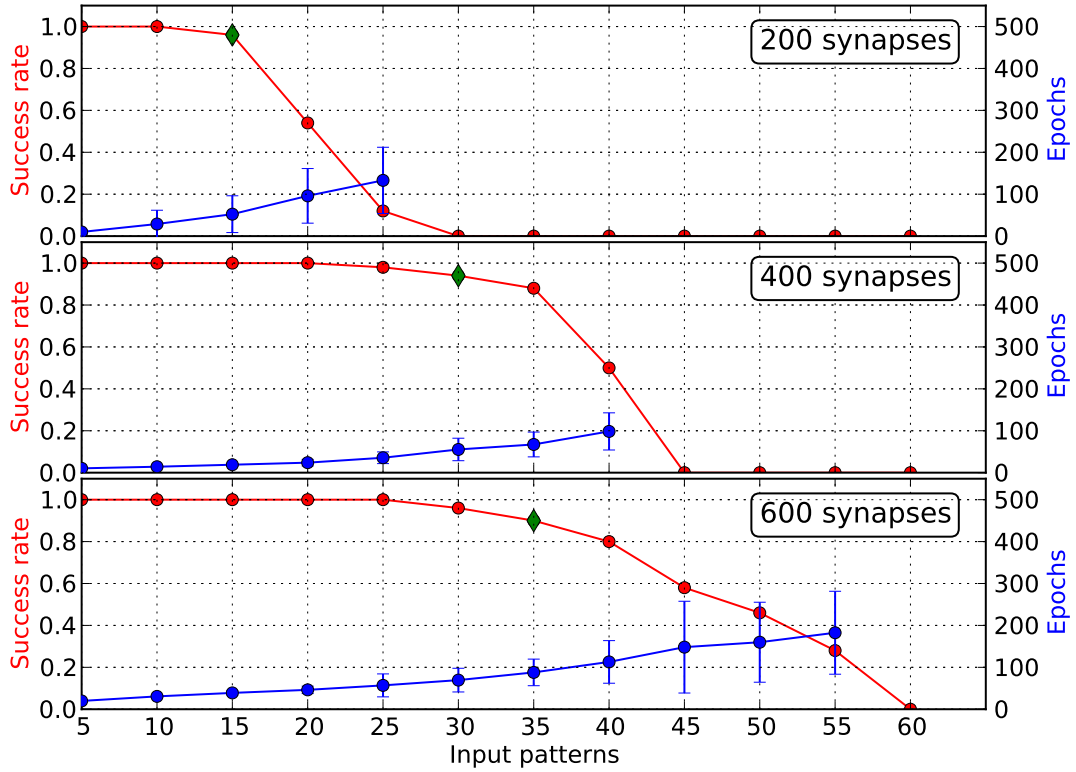
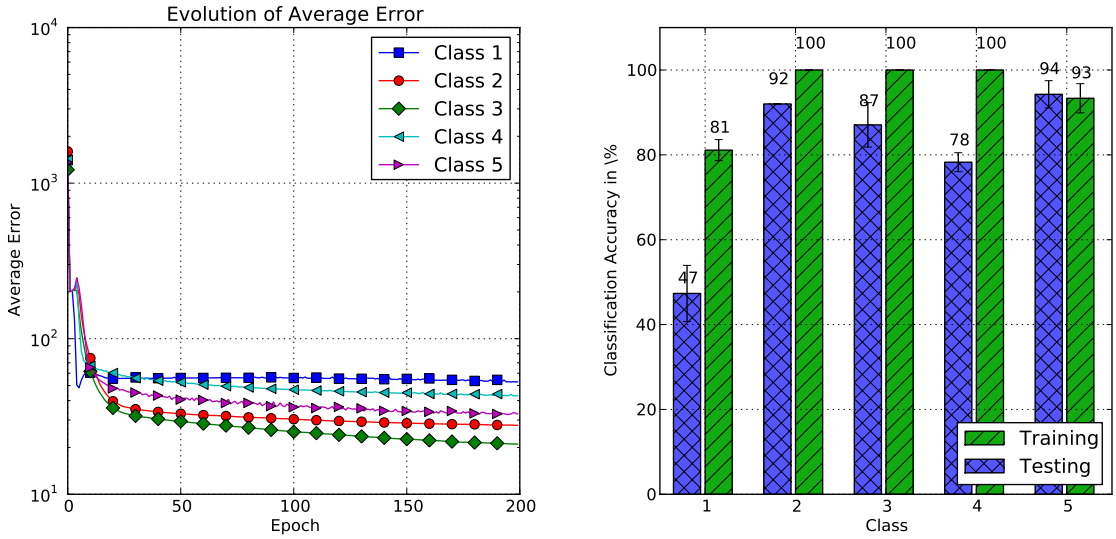


Figure 4: The memory capacity of SPAN with different number of synapses. The green diamond marker represents the maximum number of learned patterns for which the average number of successful training is above 90%.

fired spike in response to that pattern is within 2 ms of the corresponding target spike. The learning rate is set to $\lambda = \frac{5c}{p}$ and the synaptic weights are initialized randomly using maximum synaptic weights of 5, 2.5 and 2 pA for the 200, 400 and 600 synapses respectively. These values were set based on trial and error.

Figure 4 shows the results of the experiment for the three cases of the synapses. From the figure, it is clear that increasing the number of synapses increases the number of patterns that can be remembered and classified correctly. However, more epochs and more computation time is required to adjust the synaptic weights. It is noted that after a certain number of input patterns, it becomes difficult for the neuron to recognize the patterns, hence, the success rate starts to drop. We consider the points where the success rate is 90% and above, which are indicated by the green diamond markers in Figure 4. For these points, the value of p is 15, 30, 35 with success rate of 96%, 94%, 90% respectively. Furthermore, the average number of epochs to achieve successful training is below 100. The load factor at these points is computed to be 0.075, 0.075 and 0.058 for the three cases of 200, 400 and 600 synapses respectively. To get a sense of these values, We have conducted an experiment to measure the memory capacity of ReSuMe learning rule at these points, *i.e.* with the same values of p and n. For this experiment, a batch learning rule of ReSuMe was used (see Eq.20 in section 4, with the value of a_R was set to 0.025 and the learning rate was set to 10). The obtained success rates of ReSuMe to learn to recognize the input patterns were 22%, 10% and 52%. These values are lower than the success rates of SPAN, hence, ReSuMe has less memory capacity than SPAN.



(a) (b)

Figure 5: Evolution of the average errors obtained in 30 independent trails for each class of the training samples (**a**). The average accuracies obtained in the training and testing phase (**b**).

3.5. *Classification problem*

In this experiment a spatio-temporal classification task is performed. The objective is to learn to classify five classes of input spike patterns. The pattern for each class is given as a random input spike pattern that was created in a similar fashion as for the previous experiment. Fifteen copies for each of the five patterns are then generated by perturbing each pattern using a Gaussian jitter with a standard deviation of 3ms resulting in a total of $15 \times 5 = 75$ samples in the training data set. Additionally, we create $25 \times 5 = 125$ testing samples using the same procedure. The output neuron is then trained to emit a single spike at a specific time for each class. Only the training set is used during training, while the testing set is used to determine the generalization ability of the trained neuron. The spike time of the output neuron encodes the class label of the presented input pattern. The neuron is trained to spike at the time instances 33, 66, 99, 132, and 165ms respectively, each spike time corresponding to one of the five class labels. We allow 200 epochs for the learning method and we repeat the experiment in 30 independent runs. The number of synapses in this experiment was set to 200. For each run we chose a different set of random initial synaptic weights.

Figure 5a shows the evolution of the average error for each of the five classes. In the first few epochs, the value of the error oscillates and then starts to stabilize and decrease slowly. The learning error decreases for some classes faster than for others, *e.g.* class 3. We also note that the class reporting the highest error is class 1. This behavior is expected and confirms a quite similar finding in [41]. In order to classify samples of class 1 correctly, the output neuron has to emit a very early spike at $t \approx 33$ms. Consequently, the neuron needs to be stimulated by input spikes occurring at times before $t = 33$ms. However, due to the random generation of the input data, only few input spikes occur before $t = 33$ms. Most input spikes arrive after that time at the output neuron and therefore do not contribute to the correct classification of class 1 samples. The relationship between the accuracy and the output spike time was also noted in [41]. Future studies will further investigate this interesting observation.

In order to report the classification accuracy of the trained neuron, we define a simple error metric. We consider a pattern as correctly classified, if the neuron fires a single spike within $[t_d^f - 3\text{ms}, t_d^f + 3\text{ms}]$ of the desired spike time t_d^f. Any other output is

considered as incorrect. It is noteworthy to mention that using this definition, an untrained neuron is very likely to produce incorrect outputs resulting in accuracies close to zero. Figure 5b shows the average classification error for each class in the training and testing phase. As mentioned above, for testing, the 125 unseen patterns of the test set are used. The neuron is able to learn to classify the 75 training patterns with an average accuracy of 94.8% across all classes. Once more, we note the comparatively poor classification performance of samples belonging to the first class. For the test patterns, the neuron is able to achieve average accuracy of 79.6% across all classes.

4. Discussion

The experimental analysis presented in the previous section has demonstrated that, despite its algorithmic simplicity, the SPAN learning method can efficiently impose a desired input/output behavior to a SNN. In this section, we compare the differences and the similarities between the proposed method and two related algorithms, the Chronotron [41] and the ReSuMe learning rule [35,36].

Similar to SPAN, also the ReSuMe learning algorithm is derived from the Widrow-Hoff rule. ReSuMe interprets the Widrow-Hoff rule as a combination of an STDP and an anti-STDP process. With the introduction of an explicit learning window, the method emphasizes on the implementation of biologically plausible learning processes. The SPAN rule, on the other hand, follows a different idea. The sacrifice of biological realism allows the straightforward formulation of an efficient synaptic weight modification rule. By converting spike trains into analog signals, the Widrow-Hoff rule can be directly applied to spiking neurons. Despite the fact that the kernelization of spike trains was investigated in several studies before, we are not aware of any study that applies spike convolution in an algorithm for the learning of precisely timed spike train patterns. In [26,36] kernel functions have been used to define spike train metrics and in [48] kernelized spike trains were studied in the context of classification problems using a nearest neighbor approach.

Although the biological plausibility of the SPAN learning method is at least questionable, a surprising observation can be made when the α-kernel employed in this study is replaced by an exponential one. We define a convolved spike using an exponential kernel as:

$$\begin{aligned} \tilde{s}(t-t^f) &= \kappa(t-t^f) \\ &= H(t-t^f)e^{-(t-t^f)/\tau} \end{aligned} \tag{18}$$

Using this kernel, the integration of Eq. 7 leads to:

$$\begin{aligned} \Delta w_i^{\mathrm{SPAN}} &= \lambda \int_0^\infty \Delta w_i(t) \\ &= \lambda \int_0^\infty \tilde{x}_i(t)\ (\tilde{y}_d(t) - \tilde{y}_a(t))\ \mathrm{dt} \\ &= \frac{\tau\lambda}{2}\left[\sum_g \sum_f e^{-\frac{|t_d^g - t_i^f|}{\tau}} \right. \\ &\quad \left. - \sum_h \sum_f e^{-\frac{|t_a^h - t_i^f|}{\tau}}\right] \end{aligned} \tag{19}$$

This form of the SPAN learning rule has a surprising similarity to the ReSuMe rule. A batch learning version of ReSuMe was given in [41] and is defined as:

$$\begin{aligned} \Delta w_i^{\mathrm{ReSuMe}} &= \lambda \left[\sum_g \left(a_R + \sum_{t_i^f < t_d^g} e^{-\frac{t_d^g - t_i^f}{\tau}}\right) - \right. \\ &\quad \left. \sum_h \left(a_R + \sum_{t_i^f < t_a^h} e^{-\frac{t_a^h - t_i^f}{\tau}}\right)\right] \end{aligned} \tag{20}$$

where a_R is a non-Hebbian term that was shown to be important to speed up the convergence of the training process [49].

Both rules differ in the way the spikes are accumulated, *cf.* the inner sum loops over all input spikes t_i^f in Eq. 19. While ReSuMe only accumulates spikes that occur *before* an input spike t_i^f, SPAN's learning rule does not include this discrimination. Furthermore, we have noticed that SPAN can achieve better results when using an α kernel compared to an exponential one. However, it will be interesting to investigate the impact of the kernel functions on the performance of SPAN in a future study. Our preliminary comparison shows that SPAN with α kernel function has much better memory capacity than ReSuMe as reported in section . A similar observation were made in [41] when comparing ReSuMe and Chronotron.

In concept, the SPAN rule is also similar to the Chronotron E-learning rule [41]. Also in Chronotron

the synaptic weights are modified according to a gradient descent in an error landscape. Its error function is based on the Victor&Purpura (VP) distance [42]. By finding a way to deal with the discontinuities of the VP metric, the Chronotron rule efficiently computes the error gradient and updates the weights accordingly. SPAN's error landscape, on the other hand, is based on a metric similar to the van Rossum metric [43] but with α kernels. This metric does not exhibit any discontinuities allowing the definition of a simple yet powerful learning rule. A future study should investigate the differences and similarities of Chronotron and SPAN in detail.

5. Conclusion and future directions

In this paper we have proposed SPAN, a spiking neuron that uses a new learning rule to impose a desired input/output behavior to a SNN. The SPAN learning algorithm is based on the transformation of spike trains into analog signals which allows the application of the existing Widrow-Hoff rule directly for training the neuron. Although being described in many related studies, the convolution of spike trains using kernel functions was, to the best of the author's knowledge, not applied to any SNN learning method before. The conversion is performed externally during the training to compute an error signal, thus SPAN does not depend on the neural model and other models (in addition to the LIF neuron used here) could be employed as well. The algorithm was tested on different spatio-temporal tasks including a spike pattern classification problem that resembles to some extent a real world problem. We demonstrated the robustness of the learning process for noisy input patterns and showed the memory capacity of SPAN in dependence of the number of synapses. As was elaborated in the discussion section, the SPAN learning algorithm is related to other learning methods, *i.e.* ReSuMe and Chonotron, although SPAN clearly targets an application to engineering problems and is therefore less concerned with biological aspect.

Future studies will apply SPAN to spatio-temporal computation tasks such as human action recognition, gesture recognition, sound and speech recognition and the processing of EEG signals for brain-computer interfaces. Furthermore, the application of SPAN as a readout function for Liquid State Machines is straightforward. Since the here presented version of SPAN is implemented as a batch learning process, the suitability of the described online learning version will be analyzed. Additionally, the impact of stochastic neural models [50] is planned to be investigated.

Acknowledgements

The work on the paper is supported by the Knowledge Engineering and Discovery Research Institute (www.kedri.info) funded by the Auckland University of Technology, New Zealand. A. Mohemmed has been funded by the BuildIT Foundation. N. Kasabov has been also supported by the EU FP7 Marie Curie project PIIF-GA-2010-272006 EvoSpike, hosted by the Institute for Neuroinformatics – the Neuromorphic Cognitive Systems Group, at the University of Zurich and ETH Zurich.

References

1. W. Gerstner and W. M. Kistler: *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, Cambridge, MA (2002).
2. W. Maass: Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, **10**(9) (1997) 1659 – 1671.
3. W. Maass and C. M. Bishop (eds.): *Pulsed Neural Networks.* MIT Press, Cambridge, MA, USA (1999).
4. W. Maass: Computing with spiking neurons. In *Pulsed neural networks*, 55–85. MIT Press, Cambridge, MA, USA (1999).
5. S. Ghosh-Dastidar and H. Adeli: Spiking neural networks. *Int. J. Neural Syst.*, **19**(4) (2009) 295–308.
6. W. Maass and A. M. Zador: Dynamic stochastic synapses as computational units. *Neural Computation*, **11**(4) (1999) 903–917.
7. M. Tsodyks, K. Pawelzik and H. Markram: Neural networks with dynamic synapses. *Neural Comput.*, **10**(4) (1998) 821–835.
8. S. M. Bohte: The evidence for neural information processing with precise spike-times: A survey. *NATURAL COMPUTING*, **3** (2004) 2004.
9. R. VanRullen, R. Guyonneau and S. J. Thorpe: Spike times make sense. *Trends in Neurosciences*, **28**(1) (2005) 1 – 4.
10. D. A. Butts, C. Weng, J. Jin, C. Yeh, N. A. Lesica, J. Alonso and G. B. Stanley: Temporal precision in the neural code and the timescales of natural vision. *Nature*, **449**(7158) (2007) 92–95.
11. P. Tiesinga, J. Fellous and T. J. Sejnowski: Regulation of spike timing in visual cortical circuits. *Nat Rev Neurosci*, **9**(2) (2008) 97–107.
12. B. Szatmry and E. M. Izhikevich: Spike-timing the-

ory of working memory. *PLoS Comput Biol*, **6**(8) (2010) 1–11.
13. J. Hopfield: Pattern recognition computation using action potential timing for stimulus representation. *Nature*, **376** (1995) 33–36.
14. S. M. Bohte, J. N. Kok and J. A. L. Poutré: Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, **48**(1-4) (2002) 17–37.
15. J. Iglesias and A. E. P. Villa: Emergence of preferred firing sequences in large spiking neural networks during simulated neuronal development. *Int. J. Neural Syst.*, **18**(4) (2008) 267–277.
16. N. Kasabov, R. Schliebs and H. Kojima: Probabilistic computational neurogenetic modelling: From cognitive systems to alzheimers disease. *Autonomous Mental Development, IEEE Transactions on*.
17. U. R. Acharya, E. C. Chua, K. C. Chua, L. C. Min and T. Tamura: Analysis and automatic identification of sleep stages using higher order spectra. *International Journal of Neural Systems*, **20**(6) (2010) 509–521.
18. M. OHalloran, B. McGinley, R. C. Conceicao, F. Morgan, E. Jones and M. Glavin: Spiking neural networks for breast cancer classification in a dielectrically heterogeneous breas. *Progress In Electromagnetics Research C*, **113** (2011) 413–428.
19. S. G. Wysoski, L. Benuskova and N. Kasabov: Evolving spiking neural networks for audiovisual information processing. *Neural Networks*, **23** (2010) 819–835.
20. S. Ghosh-Dastidar and H. Adeli: A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Networks*, **22**(10) (2009) 1419 – 1431.
21. J.-A. Pérez-Carrasco, C. Serrano, B. Acha, T. Serrano-Gotarredona and B. Linares-Barranco: Spike-based convolutional network for real-time processing. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ICPR'10, 3085–3088. IEEE Computer Society, Washington, DC, USA (2010).
22. S. Soltic and N. K. Kasabov: Knowledge extraction from evolving spiking neural networks with rank order population coding. *Int. J. Neural Syst.*, **20**(6) (2010) 437–445.
23. S. P. Johnston, G. Prasad, L. Maguire and T. M. McGinnity: An fpga hardware/software co-design towards evolvable spiking neural networks for robotics application. *International Journal of Neural Systems*, **20**(6) (2010) 447–461.
24. E. Nichols, L. J. McDaid and N. H. Siddique: Case study on a self-organizing spiking neural network for robot navigation. *International Journal of Neural Systems*, **20**(6) (2010) 501–508. PMID: 21117272.
25. N. Kasabov: *Evolving Connectionist Systems: The Knowledge Engineering Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, second edn. (2007).
26. W. Maass, T. Natschläger and H. Markram: Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, **14**(11) (2002) 2531–2560.
27. S. J. Thorpe, R. Guyonneau, N. Guilbaud, J.-M. Allegraud and R. VanRullen: SpikeNet: real-time visual processing with one spike per neuron. *Neurocomputing*, **58-60** (2004) 857 – 864.
28. L. Bako: Real-time classification of datasets with hardware embedded neuromorphic neural networks. *Briefings in Bioinformatics*, **11**(3) (2010) 348 –363.
29. S. M. Bohte, J. N. Kok and J. A. L. Poutré: SpikeProp: backpropagation for networks of spiking neurons. In *ESANN*, 419–424 (2000).
30. D. E. Rumelhart, G. E. Hinton and R. J. Williams: Learning representations by back-propagating errors. *Nature*, **323** (1986) 533–536.
31. J. Xin and M. Embrechts: Supervised learning with spiking neural networks. In *International Joint Conference on Neural Networks, IJCNN '01*, vol. 3, 1772–1777. IEEE Press (2001).
32. B. Schrauwen and J. van Campenhout: Improving SpikeProp: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC Workshop* (2004).
33. P. Tiňo and A. J. S. Mills: Learning beyond finite memory in recurrent networks of spiking neurons. *Neural Computation*, **18**(3) (2006) 591–613.
34. R. Gutig and H. Sompolinsky: The tempotron: a neuron that learns spike timing-based decisions. *Nat Neurosci*, **9**(3) (2006) 420–428.
35. F. Ponulak: ReSuMe – new supervised learning method for spiking neural networks. *Tech. rep.*, Institute of Control and Information Engineering, Poznań University of Technology, Poznań, Poland (2005).
36. F. Ponulak and A. Kasiński: Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Computation*, **22**(2) (2010) 467–510. PMID: 19842989.
37. C. C. Bell, V. Z. Han, Y. Sugawara and K. Grant: Synaptic plasticity in a cerebellum-like structure depends on temporal order. *Nature*, **387** (1997) 278–281.
38. R. Legenstein, C. Naeger and W. Maass: What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation*, **17**(11) (2005) 2337–2382.
39. B. Widrow and M. Lehr: 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, **78**(9) (1990) 1415 –1442.
40. F. Ponulak and A. Kasinski: ReSuMe learning method for spiking neural networks dedicated to neuroprostheses control.
41. R. V. Florian: The chronotron: a neuron that learns to fire temporally-precise spike patterns.

http://precedings.nature.com/documents/5190/version/1 (2010).

42. J. D. Victor and K. P. Purpura: Metric-space analysis of spike trains: theory, algorithms and application. *Network: Computation in Neural Systems*, **8**(2) (1997) 127–164.
43. M. C. van Rossum: A novel spike distance. *Neural Computation*, **13**(4) (2001) 751–763.
44. A. Mohemmed, S. Schliebs, S. Matsuda, K. Dhoble and N. Kasabov: Optimization of spiking neural networks with dynamic synapses for spike sequence generation using pso. In *International Joint Conference on Neural Networks*. IEEE Publishing, San Jose, California, USA (2011). In Print.
45. A. Mohemmed, S. Schliebs and N. Kasabov: Method for training a spiking neuron to associate input-output spike trains. In *Engineering Applications of Neural Networks*. Springer, Corfu, Greece (2011). In Print.
46. E. Nordlie, M.-O. Gewaltig and H. E. Plesser: Towards reproducible descriptions of neuronal network models. *PLoS Comput Biol*, **5**(8) (2009) e1000 456.
47. M.-O. Gewaltig and M. Diesmann: Nest (neural simulation tool). *Scholarpedia*, **2**(4) (2007) 1430.
48. B. Schrauwen and J. V. Campenhout: Linking non-binned spike train kernels to several existing spike train metrics. *Neurocomput.*, **70** (2007) 1247–1253.
49. F. Ponulak: Analysis of the resume learning process for spiking neural networks. *Applied Mathematics and Computer Science*, **18**(2) (2008) 117–127.
50. N. Kasabov: To spike or not to spike: A probabilistic spiking neuron model. *Neural Networks*, **23**(1) (2010) 16–19.